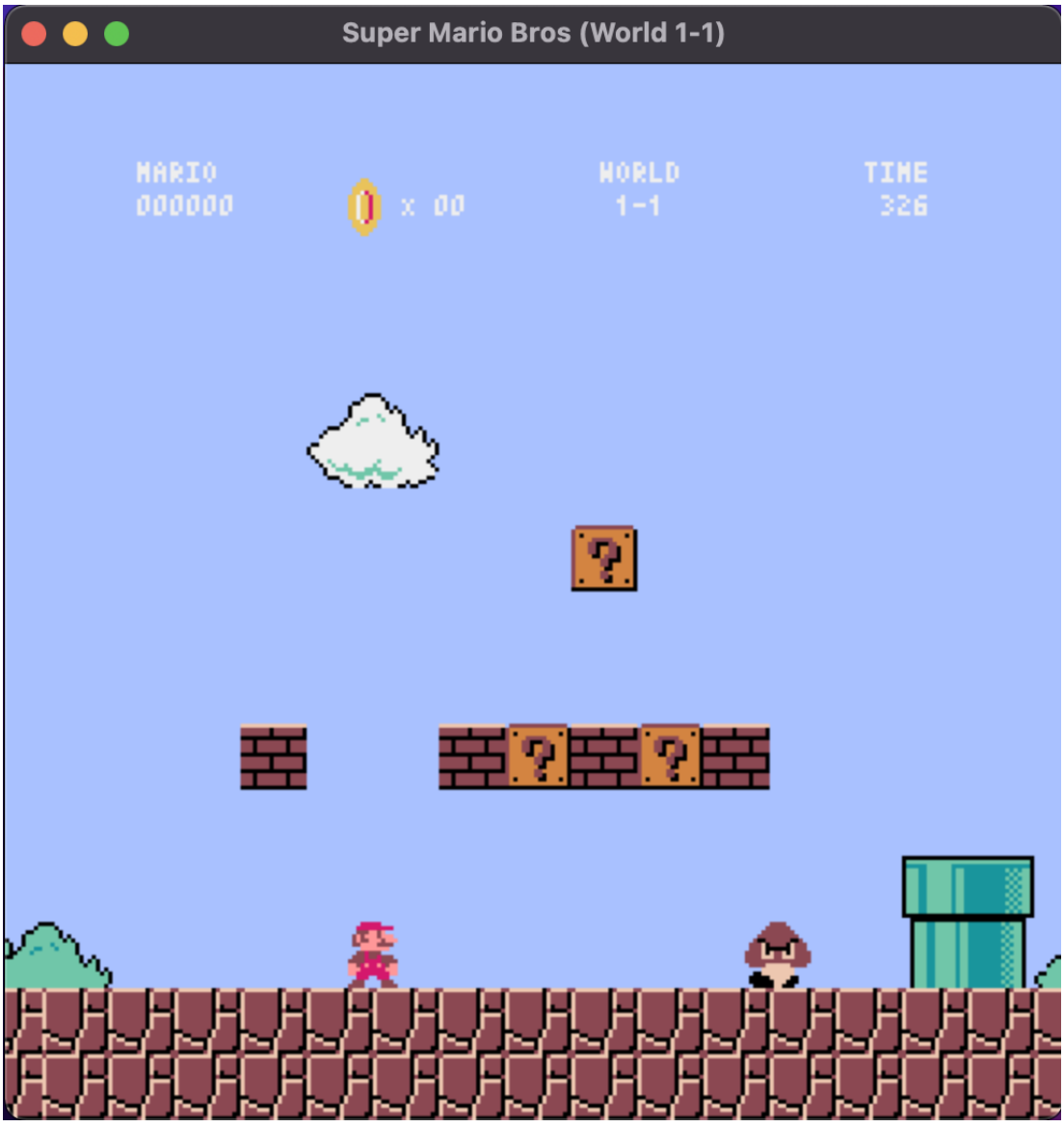
Universidad Carlos III de Madrid – Ingeniería Informática y Administración de Empresas

# Super Mario Bros

Daniel Sánchez de la Cruz

---

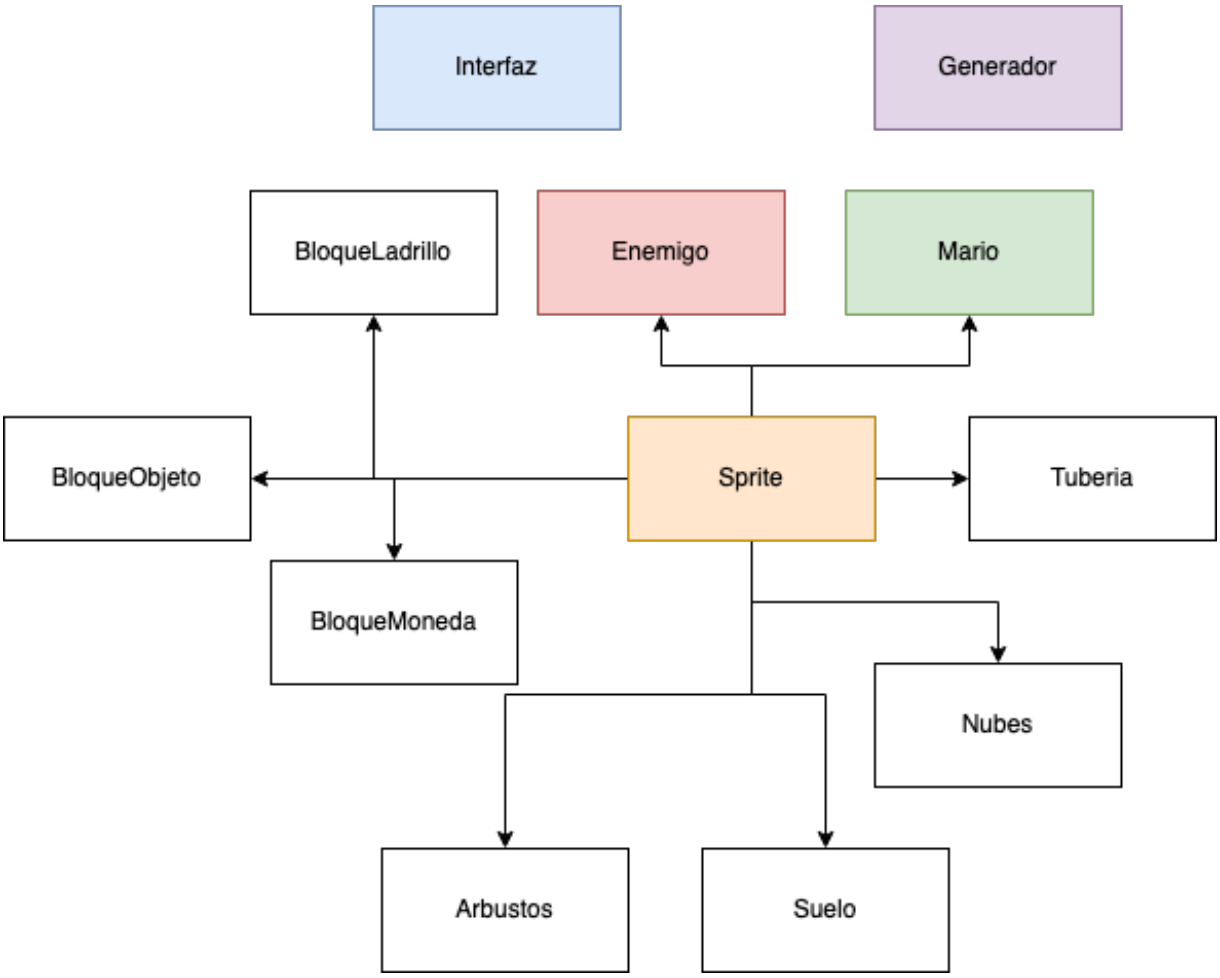


# Tabla de contenido

# Resumen

Este proyecto es una recreación del primer nivel de Super Mario Bros (videojuego original de la NES). Se ha desarrollado utilizando un motor de juegos retro llamado Pyxel. En esta memoria se recoge de manera simplificada el funcionamiento del código y una breve reflexión acerca de las dificultades encontradas a la hora de realizar el proyecto.

# Diseño de las clases




- Sprite: clase principal que heredan todos los objetos gráficos del juego
    - Atributos: utilizados para cargar en Pyxel el sprite de cada objeto
        - Self.x
        - Self.y
        - Self.img_bank
        - Self.u

        - Self.v
        - Self.w
        - Self.h
        - Self.colkey (color transparente)
    - Métodos:
        - draw(): utilizado para simplificar código y usarlo a la hora de dibujar en pantalla el objeto
        - check_collision(): utilizado para ver si dicho objeto colisiona con otros. Devuelve el objeto con el que colisiona y el lado del objeto que colisiona con él.
- Los objetos estáticos (no coloreados en el diagrama) tienen todos la misma estructura: heredan el objeto Sprite y se inicializan, cambiando únicamente su UV (localización de la imagen del objeto en el banco de imágenes).
- Mario: clase encargada de controlar el personaje principal del juego:
    - Atributos únicos:
        - Self.lives
        - Atributos utilizados para el movimiento de Mario:
            - Self.jump_time
            - Self.jump_active
            - Self.grounded
    - Métodos:
        - move(): Controla el movimiento de mario, tanto el horizontal como el salto
- Enemigo: clase que genera los enemigos del juego aleatoriamente:
    - Atributos únicos:
        - Self.enemigos[]: Una clase principal genera y guarda como atributo una lista de enemigos
        - Self.direccion: indica si el enemigo se debe dirigir hacia izquierda o derecha
    - Métodos:
        - move(): se encarga del movimiento de todos los enemigos vivos del nivel
        - generar_enemigos(): se llama constantemente para generar los enemigos en el paso del tiempo
- Interfaz: clase especial que se encarga de la puntuación, las monedas y el tiempo restante del juego, y lo imprime en pantalla.
    - Atributos:
        - Self.score
        - Self.time
        - Self.coins
        - Self.counter (utilizado para calcular el tiempo transcurrido)

    - Métodos:
        - draw(): sirve para dibujar la interfaz en pantalla
        - check_time(): Se llama constantemente para actualizar el tiempo restante y quitar una vida a Mario si llega a 0 (o cerrar el juego si no quedan vidas).
- Generador: clase auxiliar utilizada para generar listas todos los objetos estáticos del juego. Simplifica mucho el código:
    - Sin atributos.
    - Métodos:
        - Generar_objetos (necesita como entrada una lista de coordenadas del objeto a generar)
        - Dibujar_objetos

También se utiliza un archivo de constantes, donde se ubican todas las posiciones de cada objeto estático en el juego, constantes para la interfaz, los movimientos de los personajes y la física del juego, la resolución de la pantalla, los fotogramas por segundo, etc.

# Algoritmos utilizados

El único algoritmo destacable es el de la colisión. Se ha utilizado un método de colisión AABB (Axis-Aligned Bounding Box), que trata a los objetos como cajas o rectángulos, y comprueba los ejes X e Y para detectar colisiones.

# Funcionalidad incluida

- Interfaz gráfica
- Tiempo restante y vidas de Mario
- Movimiento de Mario con animaciones
- Generación aleatoria de enemigos con animaciones
- Decoración del entorno
- Interacción con bloques limitada
- Colisiones de Mario y enemigos con objetos estáticos

# Partes no realizadas

- Interacción entre Mario y los enemigos
- Objetos y monedas
- Funcionalidad extra

# Conclusiones

El proyecto ha sido complicado en ciertos aspectos, como las colisiones, que ha sido el problema que más tiempo ha llevado para implementarse, y se ha resuelto parcialmente, ya que el juego tiene algunos bugs relacionados con las colisiones verticales. Otro problema menor, que fue solucionado rápidamente, fue el desplazamiento de la cámara por el nivel.

Un videojuego como proyecto final es una opción perfecta para poner en práctica conceptos de programación orientada a objetos, pero es una tarea muy complicada (la programación en videojuegos es de las más complejas que hay) y el motor Pyxel tiene ciertas limitaciones. Aun así, me ha servido como aprendizaje.